Cassandra Tutorial

A Complete Beginner's Guide to Apache Cassandra

Prepared by

Rachit Joshi

Cassandra Tutorial

Apache **Cassandra** is a distributed NoSQL database designed for high availability, horizontal scalability, and large volumes of data. It uses a decentralized architecture in which data can be distributed and replicated across multiple nodes, helping applications continue to operate without depending on a single database server.

1. What is Cassandra?

Cassandra is an open-source distributed database originally created at Facebook and later released as an Apache project. It is designed for workloads that require high write and read throughput, fault tolerance, and the ability to scale across many machines.

2. Why Cassandra is Used

- Handles very large datasets across multiple machines.
- Supports horizontal scaling by adding nodes to a cluster.
- Provides replication for high availability and fault tolerance.
- Uses a decentralized architecture without a traditional single master.
- Offers Cassandra Query Language (CQL), whose syntax resembles SQL in many areas.
- Works well for distributed applications and workloads with predictable access patterns.

3. Important Characteristics of Cassandra

- Distributed and decentralized architecture.
- Column-oriented, wide-column data model.
- High availability through replication.
- Horizontal scalability.
- Partition-based data distribution.
- Tunable consistency for supported operations.
- Designed for failure tolerance and continuous availability.

4. Cassandra Architecture

A Cassandra cluster consists of nodes that work together to store and serve data. Data is partitioned across the cluster, and replicas can be placed on multiple nodes. Modern Cassandra deployments use a peer-to-peer architecture, so nodes can coordinate without relying on a single central master.

- Node – a machine or process participating in the Cassandra cluster.
- Cluster – a collection of Cassandra nodes working together.
- Data Center – a logical grouping of nodes, often corresponding to a physical or cloud location.
- Partition – the portion of data identified by a partition key and distributed across the cluster.
- Replica – a copy of partition data stored on another node.

5. Cassandra and NoSQL

NoSQL databases are designed for data models and access patterns that do not depend on the traditional relational table model. Cassandra uses a wide-column approach and is optimized for distributed storage and query patterns. Unlike a relational database, Cassandra data models are generally designed around the queries the application needs to perform.

6. Cassandra vs Relational Databases

Relational databases commonly emphasize normalized tables, joins, and flexible SQL queries. Cassandra instead encourages query-driven data modeling, denormalization where appropriate, and efficient access through partition keys and clustering columns.

- Relational databases commonly use tables with relationships and joins.
- Cassandra uses partitioned, distributed tables designed around application queries.
- Relational systems often scale vertically and can also scale horizontally depending on the product.
- Cassandra is designed from the beginning for horizontal, distributed scaling.
- Cassandra does not support relational joins in the same way as a traditional RDBMS.

7. Cassandra vs HBase

Cassandra and HBase are both distributed NoSQL technologies, but their architectures and operational models differ. Cassandra uses a peer-to-peer design and is known for multi-datacenter replication and high availability. HBase is closely associated with the Hadoop ecosystem and uses a different master/region-server architecture.

8. Prerequisites

A learner should have a basic understanding of databases, SQL concepts, data modeling, and command-line tools. Some familiarity with distributed systems is helpful when studying replication, consistency, and cluster architecture.

9. Installing Cassandra

Cassandra can be installed using supported packages, archives, or container-based approaches. The exact installation steps depend on the operating system and Cassandra version. After installation, the Cassandra service must be configured and started before connecting with a client.

- Install a supported Java runtime when required by the selected Cassandra version.
- Obtain Cassandra using a supported distribution or installation method.
- Configure cluster and network settings for the intended environment.
- Start Cassandra and verify that the node is running.
- Use a CQL client such as cqlsh or an application driver to connect.

10. Cassandra Query Language (CQL)

Cassandra Query Language, or CQL, is the primary language used to create and manage Cassandra keyspaces, tables, indexes, and data. Its syntax is intentionally familiar to SQL users, but CQL is designed for Cassandra's distributed data model and therefore does not provide all the features of relational SQL.

11. Keyspace

A keyspace is a top-level namespace in Cassandra. It contains tables and defines important settings such as replication strategy and replication factor. In a simple conceptual comparison, a keyspace can be thought of as roughly analogous to a database in a relational system.

```
CREATE KEYSPACE example
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 3};
```

12. Replication Factor

The replication factor specifies how many copies of a partition are maintained across the cluster. A higher replication factor can improve availability and resilience, but it also requires additional storage and coordination.

13. Replication Strategies

- SimpleStrategy – mainly intended for simple or single-data-center situations and generally not preferred for production multi-data-center deployments.
- NetworkTopologyStrategy – designed for deployments with one or more data centers and allows replication factors to be configured per data center.

14. Creating a Table

Tables in Cassandra contain columns and are organized around a primary key. The primary key determines how data is partitioned and, when a clustering component is present, how rows are ordered within a partition.

```
CREATE TABLE employee (
    id int,
    name text,
    department text,
    PRIMARY KEY (id)
);
```

15. Primary Key and Partition Key

The primary key is a critical part of Cassandra data modeling. The first component of the primary key is the partition key, which determines where the data is distributed. A composite primary key can also include clustering columns, which determine ordering within a partition.

16. Insert, Read, Update and Delete

- INSERT – adds or writes data to a table.
- SELECT – reads data from a table.
- UPDATE – changes column values for a matching primary key.
- DELETE – removes selected data.
- TRUNCATE – removes all rows from a table.

17. Cassandra Data Types

- ASCII and text/string types for character data.
- Integer and floating-point types for numeric data.
- Boolean for true/false values.
- UUID and time-related types for identifiers and timestamps.
- Blob for binary data.
- Collection types such as set, list, and map.

18. Cassandra Collections

Cassandra supports collection types for storing multiple related values within a column. The three primary collection types are set, list, and map.

- Set – stores unique elements.

- List – stores ordered elements and is useful when order matters.
- Map – stores key-value pairs.

19. Indexes

Cassandra provides indexing features for selected query patterns, but indexes should not be treated as a replacement for good data modeling. In distributed systems, query performance should be considered when deciding how data is partitioned and accessed.

20. Materialized Views

Materialized views can maintain an alternative representation of data for supported use cases. They should be evaluated carefully for the target Cassandra version and workload, with attention to operational behavior and consistency requirements.

21. Consistency Levels

Cassandra allows clients to select consistency levels for many read and write operations. A consistency level represents how many replicas must participate or respond before an operation is considered successful. The appropriate level depends on application requirements for latency, availability, and consistency.

22. Tunable Consistency

One of Cassandra's important design features is tunable consistency. Applications can choose different consistency levels for different operations instead of using a single fixed level for every request.

23. Read and Write Operations

Cassandra is optimized for predictable query patterns. Efficient reads generally identify the appropriate partition and avoid operations that require scanning a large portion of the cluster. Writes are distributed across nodes and replicated according to the configured replication strategy.

24. Cassandra Data Modeling

Cassandra data modeling begins with the application's queries. Instead of starting with a fully normalized relational model, developers identify the important access patterns and design tables that can serve those queries efficiently.

- Start with the queries the application needs.
- Choose a suitable partition key.
- Keep partitions at a manageable size.
- Use clustering columns when ordering within a partition is needed.
- Duplicate data when appropriate to avoid expensive distributed query patterns.
- Avoid designing tables primarily around relational normalization rules.

25. Cassandra Security

- Use authentication to control who can connect.
- Use authorization and roles to restrict operations.
- Protect network communication with appropriate encryption.
- Restrict access to trusted hosts and networks.
- Keep Cassandra and its dependencies updated according to operational policy.

- Monitor access and configuration changes in production environments.

26. Cassandra Backup and Recovery

A Cassandra backup and recovery strategy should account for snapshots, incremental backups, replication, cluster topology, and recovery objectives. Backups should be stored separately from the cluster and restoration procedures should be tested rather than assumed to work.

27. Cassandra High Availability

Cassandra achieves high availability through distributed nodes and data replication. If one node becomes unavailable, replicas on other nodes can continue serving data according to the configured consistency level and cluster state.

28. Scaling Cassandra

Cassandra is designed for horizontal scaling. As data volume and traffic increase, additional nodes can be added to distribute storage and workload. Capacity planning should consider data size, replication, traffic patterns, hardware resources, and the expected growth rate.

29. Common Cassandra Use Cases

- Large-scale event and activity data.
- Time-series and telemetry workloads.
- Distributed applications requiring high availability.
- High-volume write workloads.
- User activity and personalization data.
- Messaging and communication-related data.
- IoT and sensor data.
- Applications deployed across multiple geographic regions.

30. Cassandra Best Practices

- Design tables around real application queries.
- Choose partition keys carefully.
- Avoid extremely large partitions.
- Use appropriate replication strategies for the deployment topology.
- Select consistency levels according to application requirements.
- Monitor cluster health, latency, storage, and compaction behavior.
- Plan backups and test restoration procedures.
- Secure client and inter-node communication.
- Test schema and workload changes before production deployment.

31. Cassandra Learning Roadmap

- Learn basic NoSQL and distributed-database concepts.
- Understand Cassandra architecture and cluster terminology.
- Install Cassandra and connect using a CQL client.
- Learn keyspaces, tables, primary keys, and replication.
- Practice CQL CRUD operations.
- Study Cassandra data types and collections.

- Learn partition keys and clustering columns.
- Understand consistency levels and replication.
- Practice query-driven data modeling.
- Study security, backups, monitoring, and scaling.

Conclusion

Apache Cassandra is a powerful distributed NoSQL database for applications that need high availability, large-scale storage, and horizontal scalability. Its decentralized architecture, replication model, CQL interface, and query-driven data modeling make it particularly suitable for distributed workloads. A strong Cassandra foundation comes from understanding not only its commands, but also how partitioning, replication, consistency, and data modeling work together.